A Graph, Non-Tree Representation of the Topology of a Gray Scale Image

Peter Saveliev
Marshall University
One John Marshall Drive, Huntington, WV 25755, USA
saveliev@marshall.edu

Abstract

The paper provides a method of graph representation of gray scale images. For binary images, it well known that one has to capture not only connected components but also the holes. For gray scale images, there are also two kinds of "connected components" – dark regions surrounded by lighter areas or light regions surrounded by darker areas. These regions are the lower and upper level sets of the gray level function respectively. The proposed method represents the hierarchy of these sets, and the topology of the image, by means of a graph. This graph contains the well known inclusion trees but it is not a tree in general.

1. Introduction

One of the main approaches to segmentation of gray scale images relies on capturing upper and lower level sets of the gray level function of the image. The rationale for this approach is that the connected components of these sets are arguably building blocks of real items depicted in the image.

The connected components of the upper level sets have also a clear hierarchy based on inclusion. This hierarchy provides a graph representation of the topology of the image. It is called the *inclusion tree*. Various approaches to the inclusion tree have been proposed and efficient algorithms have been developed ([1], [13], [16], [17], [18] and many others).

The connected components of lower level sets may be objects and the connected components of upper level sets may be holes in these objects, or vice versa. Meanwhile, the inclusion trees for upper and lower level sets, if considered separately, do not help in finding out which object has which hole. Therefore, in order to capture the topology of the image the two trees have to be combined in some way. Approaches to such a representation are proposed in [1], [16]. However, these representations are trees.

In the present paper we follow the lower/upper level set approach to the graph representation of the topology of gray scale images but put forward a **non-tree graph representation**. The proposed graph contains complete copies of the upper and lower level inclusion trees. The proposed method is implemented via a "naïve", level-by-level, construction algorithm that simultaneously captures both lower and upper level sets in a single sweep.

The background in topology as it applies to analysis of binary images is discussed (Section 2). We also present some background for a couple of standard topological tools. The first tool is cell decomposition: the image is represented as a combination of pixels as well as edges and vertices (Section 3). The second tool is cycles: both the connected components and the holes are captured by circular sequences of edges (Section 4). The data structure – the topology graph – for the hierarchy of objects and holes is given (Section 5). The algorithm for the construction of the topology graph is incremental – every time a pixel is added, the topology of the resulting binary image is re-computed (Sections 6 and 7).

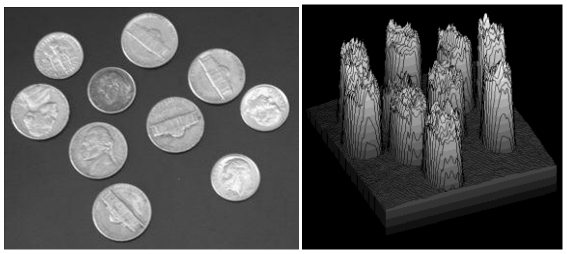
Figure 1.An image and its gray level function.

The justification for the proposed approach to the topology of gray scale images relies on both the analysis of binary images and prior research (Section 8). The graph representation of the topology – the topology graph – of a gray scale image is constructed incrementally as we go through the sequence of gray scale thresholds (Section 9). The analysis algorithm also includes filtering of the topology graph. A comparison of the topology graph to the inclusion tree, especially the version presented in [16], shows certain advantages (Section 10). An example is given of an image the topology graph of which is not a tree.

The implementation issues such as the complexity (quadratic), memory requirements (linear), and processing time of the algorithm (40 seconds per 1 million pixels) are discussed (Section 11). The conclusion is that the algorithm is practical. The algorithm has been tested and shown reasonably stable under noise and other image modifications (Section 12). Further research directions are: color images, video, and morphology (Section 13).

2. The topology of a binary image

To justify our approach to gray scale images, we will start our analysis with topology of binary images. The reason for this approach is that we prefer in the beginning to deal with topological issues in the simplest possible setting. These issues have been studied over the last 100 years or so and they are well understood.

All the background comes from [11] and [7] (in the general topological context) and from [15] (in the context of digital image analysis). However, only certain elementary concepts are used in this paper. Two basic tools are also used in the next two sections: cell decomposition and cycles.

Our goal is to capture the *topological features* present in the image: connected components and their holes.

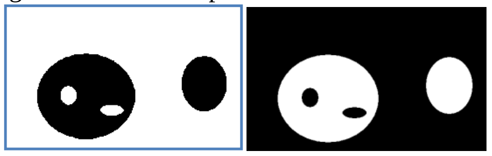

Figure 2. A binary image and its negative.

For example, in first image in Figure 2 there are two objects and the first one has two holes.

This problem is commonly known as "connected component labeling" and has many different solutions. Even though the approach we use is different from those, the output is of course the same.

We think of black objects as connected components and white objects as holes in the dark objects. However, in the second image in Figure 2 we see white objects on black background. This approach is also feasible.

To stay consistent, we have to choose one of these two options. We choose the former:

Binary images are analyzed as if they have black objects on white background.

As a result, the white objects that touch the border are not counted.

Next we make this approach more precise.

A binary image is a rectangle covered by black and white pixels arranged in a grid. In order to study the topology of the image it is reasonable to think of a pixel as a square, or a tile: [n, n + 1] × [m, m + 1].

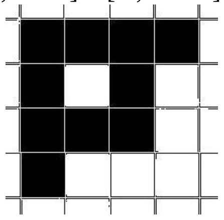

Figure 3. A binary image is represented as a combination of square tiles.

The union of any collection of pixels is a subset of the Euclidean plane. Therefore it acquires its topology from the plane [11]. In this setting, the concepts of “connected component” and “hole” have the standard meaning.

3. Cell decomposition of binary images

The boundary of a pixel is the combination of its four edges. Since an edge is shared by two adjacent pixels, keeping the list of these edges is a way to record how pixels are attached to each other. Meanwhile, keeping the list of vertices is a way to record how edges are attached to each other. This is called *cell decomposition*. The idea of cell decomposition can be traced back to the Euler formula [11].

It is standard in algebraic topology [11], [3] to represent topological spaces as “cell complexes”. This approach has been extensively applied to digital image analysis [10] and geometric modeling [14]. This method of cell decomposition differs in technical details from those that use “darts” [2], [7] (see also [12]).

In the 2-dimensional setting cells are defined as follows:

- a vertex {n}× {m} is a *0-cell*,
- an edge {n}× (m, m + 1) is a *1-cell*, and
- a face (n, n + 1) × (m, m + 1) is a *2-cell*.

Thus cells are “open”. Then, cell decomposition is a partition of the union of black (closed) pixels into the union of a collection of disjoint (open) cells.

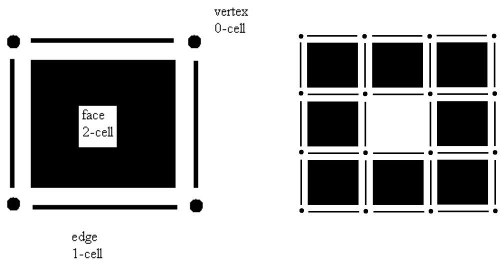

Figure 4. Left: cell decomposition of the pixel. The edges and vertices may be shared with adjacent pixels. Right: cell decomposition of the image of black eight pixels arranged in a square.

The first advantage of this approach is its generality. The pixels are attached to each other along the edges they share:

- a vertex is a 0-cell,
- two adjacent edges are 1-cells and they share a vertex, a 0-cell,
- two adjacent faces are 2-cells and they share an edge, a 1-cell.

This approach is applicable to all dimensions.

Another advantage is related to the fact that we want our algorithm to be incremental – adding one pixel at a time. As a result it can be extended to gray scale images, and later to color images, video, etc.

With cell decomposition, when we need to add a pixel to the image, we add its vertices first, then the edges, and finally the face of the pixel itself. This makes re-computing the topology easier than one that adds a whole pixel at once. Indeed, consider the fact that the new pixel is adjacent to 8 other pixels (the 8-connectedness) and these 8 pixels may belong to up to 4 different components. The result is that the number of cases to consider is quite high. Meanwhile, if an edge is added instead, the vertices are already present. The result is that there are only 4 cases to consider. The new edge may

1. link two components to each other on the outside,
2. link a component to another component inside its hole,
3. complete a hole in a component, or
4. break a hole into two.

Which of these events occurs is determined based on the local information (Figures 12-15).

The cell decomposition also makes certain geometric concepts more straightforward. First, an object and its background don't share pixels, only edges. As a result, the area of a component plus the area of the complement is exactly the total area of the image. Second, the perimeter isn't the number of pixels in its boundary but the number of edges.

At this point we provide a formal definition of the main concept of cell decomposition. Any collection γ of 0-, 1-, and 2-cells is a *cell complex* provided

- if γ contains an edge (1-cell) then γ contains its end points (0-cells) as well, and
- if γ contains a face (2-cell) then γ contains its edges (1-cells) as well.

We will refer to the cell complex of the whole rectangle as the *carrier*.

Notice that we allow a cell complex to contain vertices and edges that are not parts of faces. The reason for this is the incremental algorithm we propose adds a pixel to the image by adding its vertices, edges, and face (in that order) and we want to treat this collection of cells as a cell complex at every step.

Cell decomposition of the union of black pixels in a binary image produces a cell complex. We will call it *the cell complex of the image*.

4. Using cycles to capture topological features in binary images

Both components and holes are captured by cycles. In a cell complex,

- a *0-cycle* is any collection of vertices;
- a *1-cycle* is any collection of circular sequences of edges.

Thus a k-cycle is a collection of k-cells [11].

We will concentrate on two special kinds of cycles:

- a 0-cycle that follows the outer boundary of a connected component,
- a 1-cycle that follows the outer boundary of a hole.

We will call then *generator cycles.* As they are the only ones we will deal with, we will refer to them as simply *cycles*.

Observe that every 0-cycle of this kind can be represented by a 1-cycle, except in the case of a single vertex component. This is the approach we adopt in the following.

A 0-cycle is traversed clockwise (on the outside of the object like a rubber band) and a 1-cycle is traversed counterclockwise. In the either case black is on the right and white is on the left.

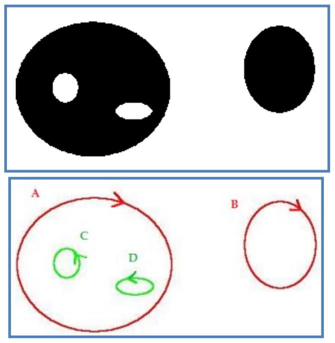

Figure 5. The objects and the holes are captured by cycles. Here A and B are 0-cycles (red), C and D are 1-cycles (green).

Given a cell complex, any cycle can be traversed by taking left turns from the initial edge in such a way that black pixels are always on the right - until this edge is reached again. This fact is used in the analysis algorithm.

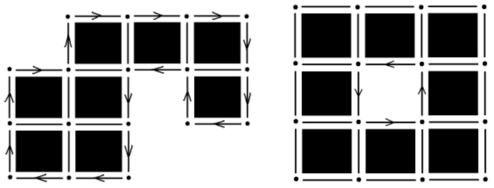

Figure 6. Left: a 0-cycle. Right: a 1-cycle.

This approach results in a natural and unambiguous representation of the regions by means of the curves that enclose them (cf. [15]).

The result of this topological analysis is a partition of the binary image. This partition is a collection of non-overlapping regions, connected sets of black pixels and connected sets of white pixels, that covers the whole image. The partition is achieved by finding boundaries of these regions as 0- and 1-cycles.

Knowing the boundaries of the objects allows us to compute their areas, moments, centroids, etc via Green's Theorem.

5. A graph representation of the topology of a binary image

First we consider a graph representation for the topology of a cell complex. It has the following simple structure. Every node represents a cycle. If an object has a hole, there is an arrow connecting these two cycles to indicate the inclusion. For convenience, we add a node for the whole rectangle (carrier). It is a direct successor of all 0-cycles.

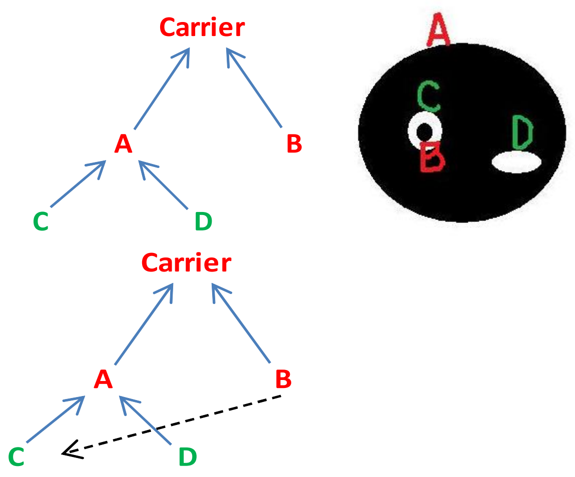

Figure 7. A graph representation of the topology of the cell complex in Figure 5, another cell complex with this representation, the topology graph of this image.

This data structure is entirely adequate to capture the topology of the cell complex. Our interest is however the whole binary image. The suggested data structure does not distinguish between an object and an object inside a hole in another object. To capture this information, we need the graph to contain this information.

The *topology graph of a binary image* is a directed graph. Its nodes correspond to the cycles (objects and holes) in the cell complex of the image and the arrows correspond to inclusions of these sets. More precisely,

- if an object has a hole, there is an arrow from the latter to the former, and
- if a hole has an object in it, there is an arrow from the latter to the former.

In addition, there is an arrow from every object to the carrier.

Thus, the topology graph captures not only the topology of the cell complex of the image but also the way this complex fits into the complex of the carrier.

The first part of the analysis algorithm is building the topology graph.

The algorithm is incremental. Starting with the empty image, pixels, one by one, are added. More precisely, a cell complex is maintained and cells are added to this cell complex as follows: vertices of the pixel first, then its edges, unless those are already present as parts of other pixels, and finally the face of the pixel. One of the alternative approaches is to add all vertices first, then all edges, then all faces.

The cell complex grows following this order:

1. the empty cell complex,
2. the cell complex of the image,
3. the carrier.

These three cell complexes will be called *frames* of the image. Besides these, the cell complex passes through intermediate stages. The cycles of the frames are called *principal cycles* and the rest are *auxiliary cycles*.

Instead of the topology graph we build the *augmented topology graph*, or simply the augmented graph, of the image. As the new pixels are added, components merge, holes split, etc. Adding a new vertex creates a new component and a new node in the graph. Adding a new edge may connect two components, or create, or split a hole. Adding the face eliminates the hole. The information about these changes is recorded in a graph. Each node in the graph represents a cycle. The directed arrows that connect the nodes represent the merging and the splitting of the cycles.

The augmented graph is an acyclic directed graph which is not a tree. The degrees are 1 or 2.

The topology graph can be extracted from the augmented graph by removing all auxiliary nodes and adding arrows between the principal nodes accordingly.

The second part of the algorithm is filtering of the principal cycles based on their characteristics such as size, location, etc.

The filtering criteria for binary images should be set in such a way that it would be impossible to remove an object while preserving a hole in it. Removing objects and holes with the area below a given threshold is such a criterion provided the area is computed as the area inside the corresponding cycle rather than the actual area of the object.

The remaining principle cycles represent the simplified topology of the image. We will call them *active cycles*.

6. Example

Before we present the pseudocode of the algorithm in the next section, we consider an extremely simple example – analysis of a single pixel image.

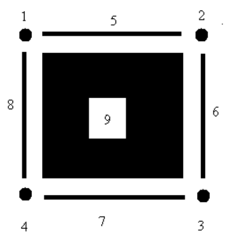

Figure 8. Adding a stand-alone pixel requires 9 steps: adding 4 vertices, 4 edges, and one face.

The process consists of 9 steps corresponding to the 9 items to be added.

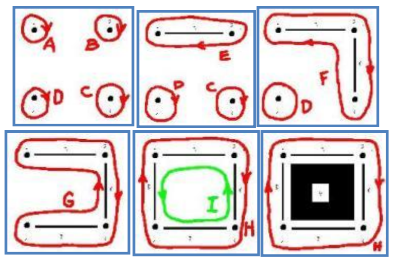

Figure 9. The stages of adding a single pixel to a blank image. The 0-cycles are in red and the only 1-cycle is in green.

The construction of the image is represented by the augmented graph. The nodes of this graph correspond to the cycles and the arrows correspond to merging and splitting of the cycles. Each arrow is accompanied by a number indicating which item is being added.

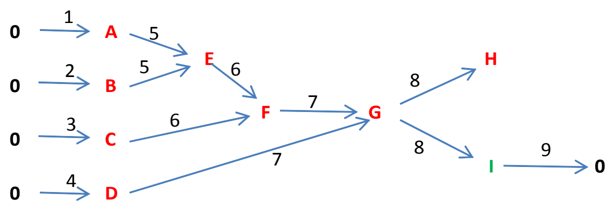

Figure 10. A graph representation of the 9 steps required to add a stand-alone pixel.

At the first stage we have 4 nodes, corresponding to the 0-cycles. Then they merge into one. This 0-cycle splits into two cycles, a 0-cycle and a 1-cycle. Finally, this 1-cycle disappears. The end result is of course just the last existing 0-cycle, H. The topology graph is H → Carrier.

Now, suppose the image consists of two black pixels on white background. Then the construction of the augmented graph starts with adding the first pixel, as above. Next, if the second pixel isn't adjacent to the first, the second stage looks exactly the same as the first. It ends with a single cycle, H'. In this case the augmented graph consists of these two parts connected to the carrier.

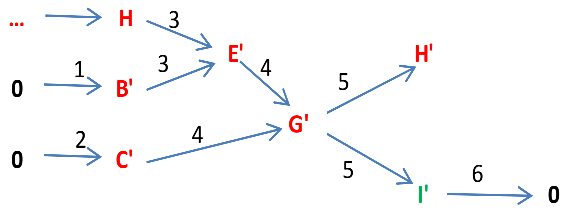

Figure 11. The augmented topology graph: adding the second (adjacent to the first) pixel to the image. This graph is attached to the graph in Figure 10.

If the second pixel is adjacent to the first, the augmented graph continues to grow beyond H. There are 6 steps. Initially only H, the 0-cycle created during the first stage, is present. Then, two new vertices are added creating two new 0-cycles, B' and C'. Then two edges are added and these three cycles merge. When another edge is added, a 1-cycle appears. Adding the face removes this cycle. Only one 0-cycle is left, H'. It corresponds to the two pixel object.

7. The pseudocode of the algorithm for binary images

A crucial part of the algorithm is the correspondence between edges and cycles. Each directed edge is given by the coordinates of its initial point and the direction: left, right, up, down. For convenience, vertices are recorded as edges with 0 direction. Then this correspondence takes the form of a table, T, so that for every directed edge E, T(E) is the cycle that passes through E. This table is updated at every step of the algorithm.

```
//---------------------------------------------
ImageAnalysis with binary image I

FOR all black pixels P in I
    CALL AddPixel with P
ENDFOR
Add all nodes with no descendants to the list
of principal cycles

FOR all white pixels P in I
    CALL AddPixel with P
ENDFOR
Add the carrier to the list of principal cycles

FOR all principal cycles C
   IF Evaluate(C) == 1 and
      Add C to the list of active cycles
   ENDIF
ENDFOR
//---------------------------------------------
```

Cycles can be evaluated based on their measurements: area, perimeter, roundness, etc. The most typical is the area, as below. Here MinArea is the lowest area set by the user.

```
//---------------------------------------------
Evaluate with cycle C

IF the area enclosed by C < MinArea THEN
   RETURN 0
```

```
ENDIF
RETURN 1
//--------------------------------------------
```

Next is the operation of adding a pixel. It includes adding its vertices, its edges, and then the face of the pixel. Adding an edge means assigning cycles to both of the directed edges – forward E and backward -E. After all the edges have been added, there is always a 1-cycle inside the pixel. It is "removed" as the face closes the hole.

```
//--------------------------------------------
AddPixel with pixel P

CALL AddVertex with upper right vertex of P
CALL AddVertex with upper left vertex of P
CALL AddVertex with lower right vertex of P
CALL AddVertex with lower left vertex of P

CALL AddEdge with lower edge of P
CALL AddEdge with right edge of P
CALL AddEdge with upper edge of P
CALL AddEdge with left edge of P

E = lower edge of P directed counterclockwise
CALL RemoveCycle with 1-cycle A = T(E)
//--------------------------------------------
```

Adding a vertex is trivial. It creates one new 0-cycle represented by a node that isn't connected to anything yet. But first you verify that the vertex isn't already present.

```
//--------------------------------------------
AddVertex with vertex V

IF V is present THEN
 RETURN
ENDIF
Mark V as present
Call CreateCycle with V RETURNING 0-cycle A
//--------------------------------------------
```

Adding an edge is the most complex step. There are four cases illustrated in Figures 12-15. Which case occurs is determined following the information in the correspondence table T.

```
//--------------------------------------------
AddEdge with edge E

IF T(E) != NULL or T(-E) != NULL THEN
 RETURN
ENDIF

CALL NextEdge with E RETURNING edge E1
A = T(E1)
CALL NextEdge with -E RETURNING edge E2
B = T(E2)

IF A == B THEN
 CALL SplitCycle with E1, E2, and A
ELSE
 CALL MergeCycles with E1 and A, B
ENDIF
//--------------------------------------------
```

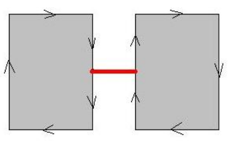

Figure 12. Case (a): the new edge connects two different 0-cycles.

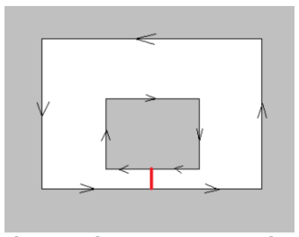

Figure 13. Case (b): the new edge connects a 1-cycle and a 0-cycle.

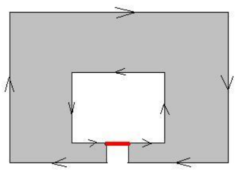

Figure 14. Case (c): the new edge connects a 0-cycle to itself.

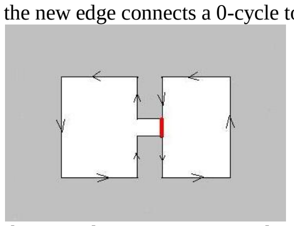

Figure 15. Case (d): the new edge connects a 1-cycle to itself.

A 0-cycle can merge with either 0- or 1-cycle: case (a) or (b).

```
//--------------------------------------------
MergeCycles with cycles A, B and edge E

CALL CreateCycle with E RETURNING cycle C

CALL MarkEdges with E and C

Add arrows from A, B to C to the graph
//--------------------------------------------
```

Either a 0- or 1-cycle can split: case (c) or (d).

```
//--------------------------------------------
SplitCycle with edges E1, E2 and cycle A

CALL CreateCycle with E1 RETURNING cycle C
CALL CreateCycle with E2 RETURNING cycle D

CALL MarkEdges with E1 and C
CALL MarkEdges with E2 and D

Add arrows from A to C, D to the graph
//--------------------------------------------
```

Creating a cycle means adding a new node to the graph.

```
//--------------------------------------------
CreateCycle with edge E

Create node A in the graph
T(E) = A
RETURN A
//--------------------------------------------
```

Removing a cycle means assigning NULL to all of its edges.

```
//--------------------------------------------
```

```
RemoveCycle with cycle A

FOR all edges E in I
   IF T(E) == A THEN
     T(E) = NULL
   ENDIF
ENDFOR
//---------------------------------------------
```

Given an edge of a cycle, one can find the next edge of the cycle.

```
//---------------------------------------------
NextEdge with edge E

Start points of edges E1, E2, E3, E4 = end point
of E
Direction of E1 = direction of E + 90 degrees
Direction of E2 = direction of E
Direction of E3 = direction of E - 90 degrees
Direction of E4 = - direction of E

FOR edge G = E1, E2, E3, E4
   IF T(G) != NULL
      RETURN G
   ENDIF
ENDFOR
//---------------------------------------------
```

Next function goes around a given cycle and assigns its value to the edges.

```
//---------------------------------------------
MarkEdges with edge E and cycle A

CALL NextEdge with edge E RETURNING edge G
WHILE G != E
   T(G) = A
   CALL NextEdge with edge G RETURNING edge G
ENDWHILE

IF the full turn is clockwise THEN
 A is a 0-cycle
ELSE
 A is a 1-cycle
ENDIF
//---------------------------------------------
```

8. The topology of a gray scale image

In a gray scale image every pixel is associated a number indicating the gray level. These numbers run from 0 to L-1, where L is usually 256 (L=2 for binary images). One can also think of it as a function of two variables, the gray level function, defined on a rectangle [0, N) × [0,M).

Even though topology of binary images is well understood [11], [7], [15], the methods are not directly applicable to gray scale images. The method suggested above however is fully applicable with minor changes.

The *gray level function* is a function from a rectangle to the set {0, 1,…, L-1} which is constant on each square [n, n + 1) × [m, m + 1).

Given a number T, *thresholding* replaces all the pixels with gray level lower than or equal to T with black leaving the rest white.

The output of thresholding is a binary image. However, because of the loss of information during thresholding, this binary image cannot always serve as an adequate replacement of the original gray scale image. Below are two examples of images for which a single threshold wouldn't work.

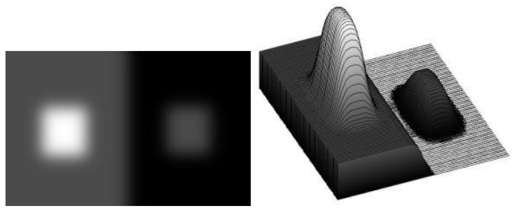

Figure 17. A gray scale image and its gray level function. The gray on the left is the same as on the right. Therefore no single threshold will capture both squares.

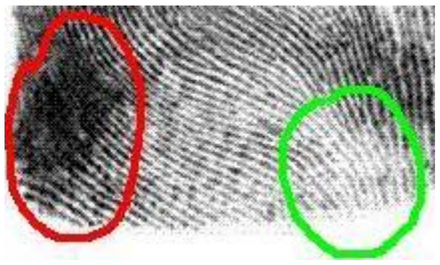

Figure 18. The gray of the ridges inside the green circle is close to the gray between the ridges inside the red circle. Therefore no single threshold will capture all ridges.

We justify our approach to analysis of the topology of gray scale images in two ways. First we appeal to analysis of binary images in the previous sections and second to prior research on the subject.

In binary images objects are either connected black areas surrounded by white background or connected white areas surrounded by black background. Similarly, our initial assumption about gray scale images will be that objects are either darker areas surrounded by lighter background or lighter areas surrounded by darker background. We propose the following terminology:

- *a dark object* is a connected component of a lower level set and
- *a light object* is a connected component of an upper level set of the gray level function.

This approach is in agreement with the following gestalt principle (Werthheimer's contrast invariance principle) [6]:

> *Image interpretation does not depend on actual values of the gray levels, but only their relative values.*

This principle suggests that one should look at the level sets of the gray scale function, as well as lower and upper level sets.

A related principle is discussed in [16]:

We assume our sensor is such that each pixel knows only if it is brighter, equal, or darker than its neighbor pixels, and that these comparisons can be propagated.

The conclusion is that the lower and upper level sets of the gray scale function should serve as the main building blocks in image segmentation. This conclusion is justified by the analyses conducted in [1], [16], [7].

As the objects are the lower and upper level sets, the boundaries of the objects are the level curves of the gray level function. Since all of these sets are connected collections of pixels they can be represented as 0- and 1-cycles, as before.

At this stage the analysis does not have unambiguous results. Let's consider the image below as an example.

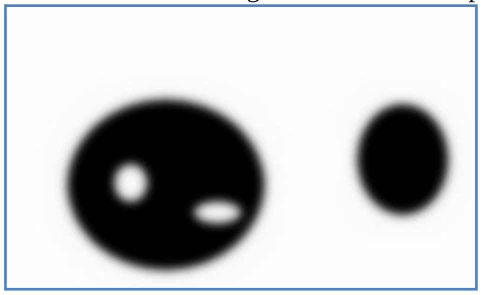

Figure 19. The blurred version of the image in Figure 5.

It may seem clear that in the above image there are two objects and the first one has two holes. However, one may choose different thresholds for different level curves. The choice of these thresholds may affect the topology of the resulting image, as illustrated in Figure 20 below. All three outcomes are equally valid.

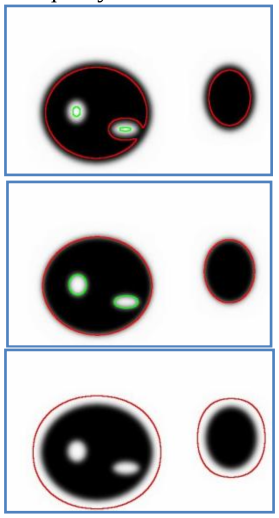

Figure 20. Each choice of thresholds produces a different topology. First: two dark objects, one with a hole – light object, another light object is a hole in the background. Second: two dark objects, one with two holes – light objects. Third: two dark objects with no holes.

Therefore the analysis has to include the following two stages:

Stage 1. Collect all possible objects in all binary images obtained via thresholding for every value of T from 0 to L. We will call these images *frames.*

Stage 2. Filter these objects to resolve the ambiguity. The result is the list of *active cycles.*

To ensure that the ambiguity is resolved properly we will adhere to the following "2D uniqueness principle":

If a dark object contains another dark object, only one of them is taken into account.

A similar principle is used for light objects. Both are vacuously true for binary images. This principle guarantees that the active cycles you cross (from inside out or from outside in) will alternate: 0-cycle, 1-cycle, 0-cycle, 1-cycle…

The justification for this principle lies in the fact that in 2D there can be no object in front of or behind another. So, either the larger object is the background for the smaller one or the smaller object is noise over the larger one. An object can have a hole in it however.

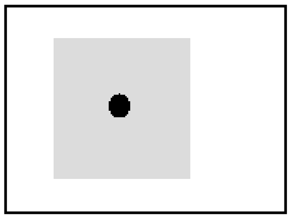

Figure 21. The circle may be noise (small size) or the square may be background (low contrast).

Stage 2 may include applying user's criteria of what is and what is not important in the image. Some specific criteria are discussed in Section 9.

The result of the analysis is an image segmentation: the active cycles partition the image into regions each of which is identified as one of the following:

- a dark object,
- a light object, or
- the background.

This segmentation satisfies the following properties:

- The outside border of a dark object is a 0-cycle. On the outside lies either a light object or the background.
- The outside border of a light object is a 1-cycle. On the outside lies either a dark object or the background.

9. A graph representation of the topology of a gray scale image

The collection of all possible cycles is organized in a graph. Its structure is very similar to that of the topology graph of a binary image. Indeed, the collections of cycles of the frames are arranged in layers within the graph. Each of these cycles encircles a dark object from the outside or a light object from the inside.

The nodes of the *topology graph of a gray scale image* correspond to light and dark objects of the image and the arrows indicate inclusions. More specifically,

- there is an arrow from every dark (light) object to a dark (light) object that contains it if they correspond to consecutive gray levels,
- there is an arrow from every dark (light) object to a light (dark) object that contains it if they correspond to the same gray level.

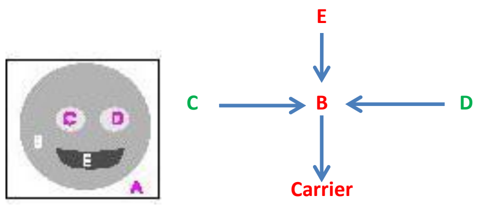

Figure 22. An image and its topology graph (L=3). Here the objects are: mouth E (black), head B (gray), eyes C and D (light gray).

The topology graph may be a tree, as in Figure 22, but generally it isn't (Figure 23).

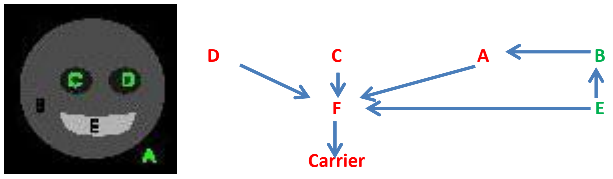

Figure 23. An image and its topology graph, which is not a tree. Here A is the complement of the face and F is the complement of the mouth.

The proposed algorithm is incremental: the layers are added one at a time. Every time we increase the threshold for the upper and lower level sets, there are six kinds of events that can (along with their combinations) happen to the connected components of these sets:

1. a dark object grows,
2. a light object shrinks,
3. a dark object appears,
4. a dark object forms a hole (a light object) inside,
5. two dark objects merge,
6. a light object splits.

The last three events change the topology of the image (see Figures 12-15). Recorded in the topology graph, these events are the reason why it is not a tree.

Just as with binary images, instead of the topology graph we build the *augmented topology graph*, or simply the augmented graph, of the image. The growing threshold creates a partial order on the set of pixels. In that order the pixels are added to the image. The procedure of building this graph with nodes representing cycles is exactly the same as the one presented in Section 5. The frames generate *principal cycles* and the rest are *auxiliary cycles*. The topology graph can be extracted from the augmented graph by removing all auxiliary nodes and adding arrows between the principal nodes accordingly.

Let's consider an example. Suppose the image consists of two adjacent pixels, black and gray, on white background. Then the construction of the augmented graph is exactly the same as described in Section 6. The topology graph is simply H → H' → Carrier. Now, H' contains H. Therefore only one of them should be active. Which one? Depending on the settings, the larger object (H') may be preserved or the one that is more round (H).

The algorithm for binary images is the process of adding pixels one by one while keeping track of changes in the topology. The same approach applies to gray scale images. In fact all pixels in the image are added in either case. The operation of adding a pixel and all functions it calls are exactly the same as in the case of a binary image (Section 7). The main difference is that there are L+1 frames instead of 3.

As some of the principal cycles are discarded, the remaining ones will represent the simplified topology of the image. If the uniqueness principle stated in Section 8 is followed, the output of the filtering step is a kind of topological sorting of the graph. Examples of image simplification based on filtering of lower and upper level sets are presented in [16] and [1].

```
//---------------------------------------------
ImageAnalysis with gray scale image I

FOR all thresholds T = 0, …, L-1
   FOR all pixels P in I
          IF the value of P <= T THEN
                   CALL AddPixel with P
          ENDIF
   ENDFOR
   Add all nodes with no descendants to the list
  of principal cycles
ENDFOR

FOR all principal cycles C
    IF Evaluate(C) == 1 and C does not have a
    direct predecessor D such that C and D are
    both k-cycles and Evaluate(D) == 1 THEN
      Add C to the list of active cycles
   ENDIF
ENDFOR
//---------------------------------------------
```

In this particular implementation of the algorithm, the objects are filtered based on their areas: objects with areas smaller than a certain threshold are considered noise. Since we measure the area of the inside the cycle that surrounds the object, it is easy to satisfy the uniqueness principle stated in Section 8. Indeed, since every successor of an object has higher area, we only need to check object's direct predecessors. Other measurements that can be used in such a simple fashion are the total intensity, the integrated optical density and similar.

What if the area is replaced with contrast? By the contrast of a dark object, we understand the difference between the highest gray level adjacent to the object and the lowest gray level within the object (similar for light objects). Unlike the area, the contrast can go up and down as we move from a lower level set to the next. Therefore the filtering step cannot be carried out by checking only the direct predecessors and successors, as with the area, and a whole graph search may be necessary. Notice that the contrast of an object is exactly its persistence [8] with respect to the sequence of frames.

Objects can be evaluated and filtered based on any combination of their measurements and other characteristics such as location. Many filtering criteria have been proposed (e.g. [4], [9], [18]). All of them are also applicable to filtering the topology graph.

10. Comparison of the topology graph to tree representations

Other approaches to graph representation of gray scale images based on the lower and upper level sets have been proposed in [1], [16] (and for 3D simplicial meshes in [5]). Below we compare the topology graph to these two representations. Both are trees.

First consider the method in [16]. Suppose we change the threshold from 0 to 255. Then, the connected components of lower level sets form a tree of growing sets. Similarly, the components of the upper level sets form a tree of shrinking sets. In either case, there are layers in the tree corresponding to the gray levels.

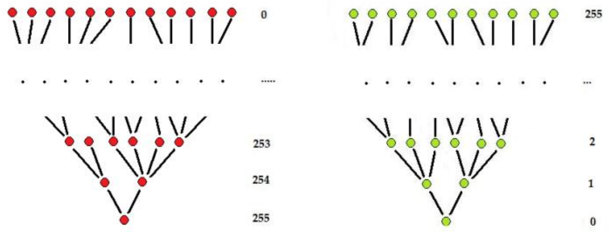

Figure 24. This diagram illustrates the lower inclusion tree and the upper inclusion tree. The red are the lower level sets (dark) and the green are the upper level sets (light). In both trees the layers correspond to the levels of gray.

The goal is to combine the two trees. The answer suggested in [16] is to consider the tree formed by connected components of the level sets. As levels sets correspond to lower and upper level sets, the end result is the "merged inclusion tree".

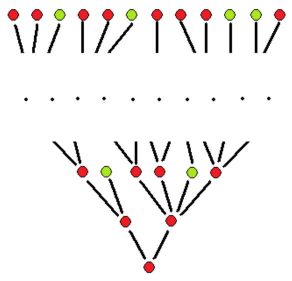

Figure 25. This diagram illustrates the merged inclusion tree. In this tree there are no layers corresponding to the levels of gray anymore.

Because of the way they are merged, the structures of the two inclusion trees are lost. In particular, the lower level sets are mixed with the upper level sets. In addition, the gray levels are also mixed. Observe that losing this information makes it impossible to filter nodes by checking only the direct predecessors. Of course, the algorithm can be modified to preserve the relations between the lower level sets and the relations between upper level sets. The result would be the topology graph.

In fact, the topology graph can be seen as an alternative way of combining the lower and upper inclusion trees. The latter is turned upside down and attached to the former.

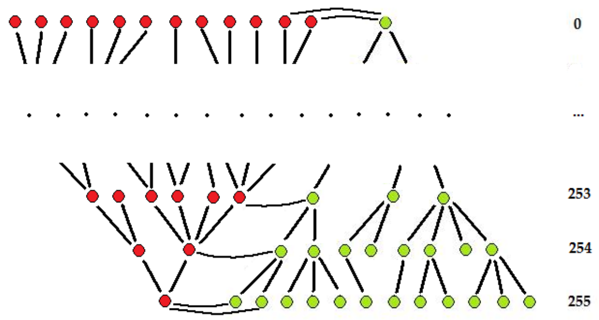

Figure 26. This diagram illustrates the structure of the topology graph of a gray scale image. Only some of the links connecting the nodes corresponding to the same gray level are shown.

To summarize, the structure of the topology graph of a gray scale image differs from that of the merged inclusion tree in a number of ways:

- The lower and upper inclusion trees remain intact within the graph.
- The graph breaks into layers that coincide with of the topology graphs of the corresponding frames.
- The topology graph is not a tree in general.

In [1], the tree structure appears as morphological openings and closings of larger and larger scale are applied to the image. In other words, the minima and maxima of the gray scale function are flattened - slice by slice. The result is a hierarchy of lower and upper level sets that is recorded as a "scale tree". Their example is below.

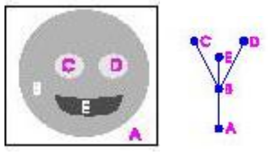

Figure 27. An image and its scale tree. Its topology graph is in Figure 22.

11. Complexity of the algorithm

Suppose N is the number of pixels in the image.

The memory usage is O(N). Indeed, there are two main data structures: cycles and edges. As adding a pixel creates no more than 9 nodes in the augmented graph, there are O(N) cycles. There are also O(N) edges in the image.

During the construction of the topology graph, each of the N pixels is processed separately and each time at most 9 new objects are created. The edges of the boundaries of some of these objects are marked, which can be done in O(N). Therefore, the complexity of this part of the algorithm is at most O(N^2).

As we go through the levels of gray in the image in Figure 28, the objects grow and fill the image. The boundary of each has to be traversed. Then the total number of edges to be visited is, roughly, 1+2+3+…+N, which adds up to O(N^2). Therefore, the complexity of the algorithm is in fact O(N^2).

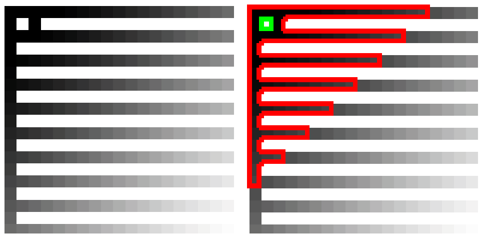

Figure 28. Left: the "comb". Its perimeter is O(N). Right: one of its lower level sets.

Such an image may seem unusual but images of maps and microchips may fall into this category. Images of cells or other particles do not.

Suppose the image in Figure 28 is not quantized. Then the above analysis of the complexity of the algorithm applies to any other algorithm that traces all level sets in the image. This is the case with the fast level line transform [16] (step 4a). Therefore its complexity is O(N^2), not O(NlogN) as claimed.

Regardless of the design of the algorithm, tracing the level sets is necessary in order to compute the perimeters of lower and upper level sets. Only then they can be filtered based on their shapes.

The complexity of filtering the graph is O(N).

The algorithm was tested with a variety of images. The processing included:

- constructing the topology graph,
- computing various measurements of all objects,
- filtering the objects based on these measurements and user's settings.

The testing was done on HP Pavilion laptop with Intel Core 2 Dual CPU T7500 2.2GHz.

The diagram below provides an estimate for the processing time for natural images of reasonable size.

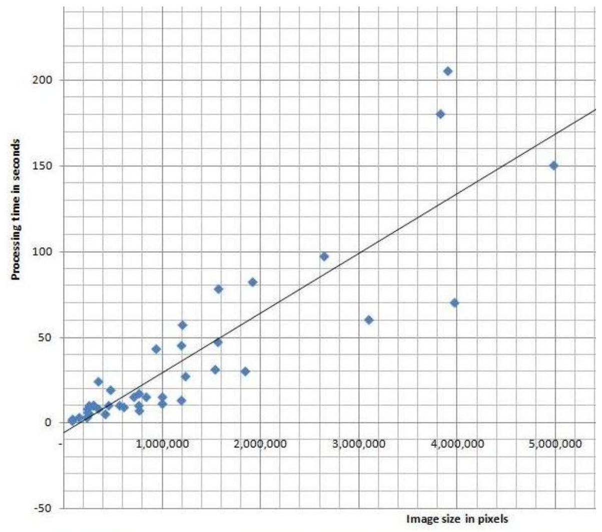

Figure 29. The processing time as a function of the image size.

For this range of sizes, the dependence appears close to linear. In this case, the processing takes, roughly, 40 seconds for each million pixels in the image. The results suggest that the algorithm is practical.

The performance of the algorithm presented in this paper is sacrificed for the sake of simplicity. One can expect that faster algorithms will be developed.

12. Experiments

To further demonstrate the practicality of the algorithm, we analyze a few simple images.

During processing, the areas, perimeters, and contrasts of the objects are also computed. These parameters are used to filter objects. The user indicates ranges of parameters of the features he considers to be irrelevant or noise.

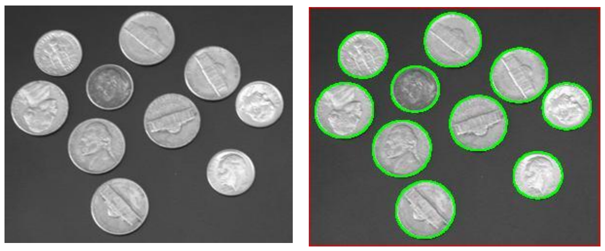
Figure 30. The coins are captured.

The settings may be chosen based on *a priori* knowledge about the image. For example, the image in Figure 30 is 300×246. To capture the coins and ignore the noise and the small features depicted on the coins, one sets the lower limit for the area at 1000 pixels.

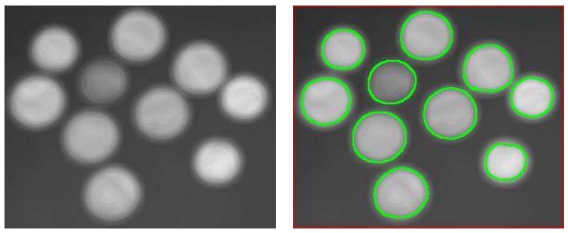
Figure 31. The coins are captured in the blurred version of the coins.

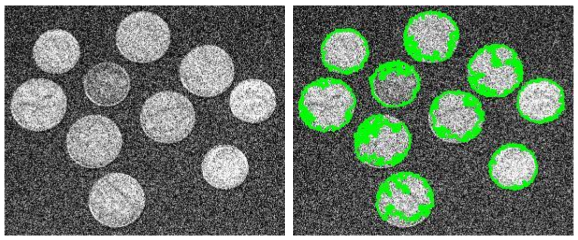
Figure 32. The coins are captured in the version of the coins with salt-and-pepper noise.

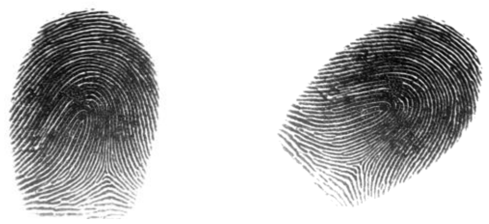
Figure 33. The image of a fingerprint and its rotated version.

A good test of robustness of an image analysis method is the degree of its stability under rotations. The output for the original 640×480 fingerprint in Figure 33 is 3121 dark and 1635 light objects. For the rotated version, it is 2969-1617. The errors are about 5% and 1%. By limiting the analysis to objects with area above 50 pixels, the results are improved to 265-125 and 259-124, respectively (2% and 1%). This test shows that even though the algorithm is sensitive (as is the image itself) to rotations, the error is reasonable and decreases as the objects get larger.

An example of image simplification based on filtering of the topology graph is in Figure 34, below.

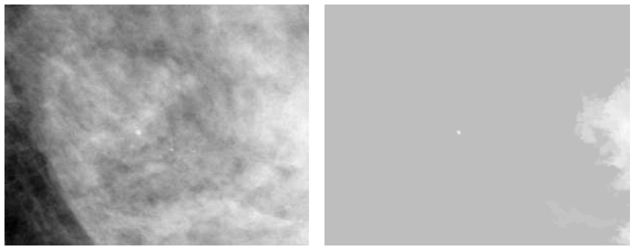
Figure 34. An MRI of breast tissue and its version with all low contrast objects removed.

Stretching the image does not affect the count of objects. Shrinking makes objects merge. If the goal, however, is to count and analyze larger features, limited shrinking of the image does not affect the outcome. The count is also stable under noise and blurring.

The method works best with images that represent something 2-dimensional. It is not applicable to the images for which the third dimension is essential. For example, the method fails when the image contains:

- occluded objects,
- transparent objects,
- objects well lit on one side and dark on the other.

The method does not incorporate any morphological operations. As a result, scratches can’t be repaired or clustered cells can’t be separated unless there is a variation of intensity.

13. Conclusions and further research

The method proposed in this paper is well grounded in classical mathematics and produces meaningful results for various gray scale images. Its speed and memory requirements make it practical for every-day use on today’s personal computers.

One of the proposed innovations is that the topology of a gray scale image is represented by a graph, which is not a tree in general.

The main issue still to be addressed is the issue of merging objects based on similarity and proximity rather than relative gray levels.

A modification of the algorithm applies to color images and other multi-parameter images. It is possible to threshold an RGB image so that a binary image (frame) is created for each combination of red, green, and blue. The cycles from these frames form the topology graph of the color image.

The ability to remove pixels in addition to adding pixels makes the algorithm track objects in a binary video. The same approach is applicable to sequences of morphological operations applied to a given binary image.

To develop a similar method for representation of the topology of 3D (and higher-dimensional) images one will follow the same pattern: decompose the image into 0-, 1-, 2-, and 3-cells, then capture the topology with 0-, 1-, 2-, and 3-cycles, combine them into homology classes, and record the hierarchy of these classes as a graph.

The following is a summary of characteristics of the proposed approach:

- The approach and the method are justified by appealing to classical mathematics.
- The new representation of the topology of a gray scale image ensures that components and holes are treated in a unified way and yet kept separate.
- The new algorithm and its interpretation are simple and easy to understand.
- The algorithm is practical.
- The algorithm is incremental and as such can be easily generalized to analysis of color images and video.

References

[1] J. A. Bangham, J. R. Hidalgo, R. Harvey, and G. C. Cawley, The segmentation of images via scale-space trees, *British Machine Vision Conference*, pp. 33-43, 1998.

[2] Y. Bertrand, C. Fiorio, and Y. Pennaneach, Border map: a topological representation for nD image analysis. In *8th International Conference on Discrete Geometry for Computer Imagery* (DGCI'1999), Marne-la-Vallée, France, Springer, Lecture Notes in Computer Science, Vol. 1568, pp. 242-257, 1999.

[3] G. Bredon, *Topology and Geometry*, Springer, 1993.

[4] E. J. Breen and R. Jones, Attribute openings, thinnings, and granulometries, *Computer Vision and Image Understanding*, Vol. 64, no. 3, pp. 377-389, 1996.

[5] H. Carr, J. Snoeyink, and U. Axen, Computing contour trees in all dimensions, *Computational Geometry*, 2003.

[6] A. Desolneux, L. Moisan, and J.-M. Morel, *From Gestalt Theory to Image Analysis, A Probabilistic Approach*, Springer, 2007.

[7] G. Damiand, Y. Bertrand, C. Fiorio, Topological model for two-dimensional image representation: definition and optimal extraction algorithm, *Computer Vision and Image Understanding*, Vol. 93, no. 2, pp. 111-154, 2004.

[8] H. Edelsbrunner, D. Letscher, and A. Zomorodian, Topological persistence and simplification. *Discrete Comput. Geom.*, Vol. 28, pp. 511-533, 2002.

[9] R. Jones, Connected filtering and segmentation using component trees, *Computer Vision and Image Understanding*, Vol. 75, no. 3, pp. 215-228, 1999.

[10] T. Kaczynski, K. Mischaikow, and M. Mrozek, *Computational Homology*, Appl. Math. Sci., Vol. 157, Springer, 2004.

[11] L. C. Kinsey, *Topology of Surfaces* (Undergraduate Texts in Mathematics), Springer, 1997.

[12] R. Klette and A. Rosenfeld, *Digital Geometry*. Morgan Kaufmann, 2004.

[13] S. Kundu, Peak-tree: a new approach to image simplification, *Proc. SPIE Vision Geometry* VIII, Vol. 3811, p. 284-294, 1999.

[14] P. Lienhardt, Topological models for boundary representation: a comparison with n-dimensional generalized maps, *Computer-Aided Design*, 23(1), pp. 59-82, 1991.

[15] R. Malgouyres and M. More, On the computational complexity of reachability in 2D binary images and some basic problems of 2D digital topology, *Theoretical Computer Science*, Vol. 283/1, no 1, pp 67-108, 2002.

[16] P. Monasse and F. Guichard, Fast computation of a contrast invariant image representation. *IEEE Transactions on Image Processing*, 9(5), pp. 860–872, 2000.

[17] L. Najman and M. Couprie, Building the component tree in quasi-linear time, *IEEE Transactions on Image Processing*, Vol. 15, no. 11, pp. 3531-3539, 2006.

[18] P. Salembier, A. Oliveras and L. Garrido, Antiextensive connected operators for image and sequence processing, *IEEE Transactions on Image Processing*, Vol. 7, no. 4, pp. 555-570, 1998.

[19] P. Salembier and J. Serra, Flat zones filtering, connected operators, and filters by reconstruction, *IEEE Transactions on Image Processing*, Vol. 4, no. 8, pp. 1153-1160, 1995.